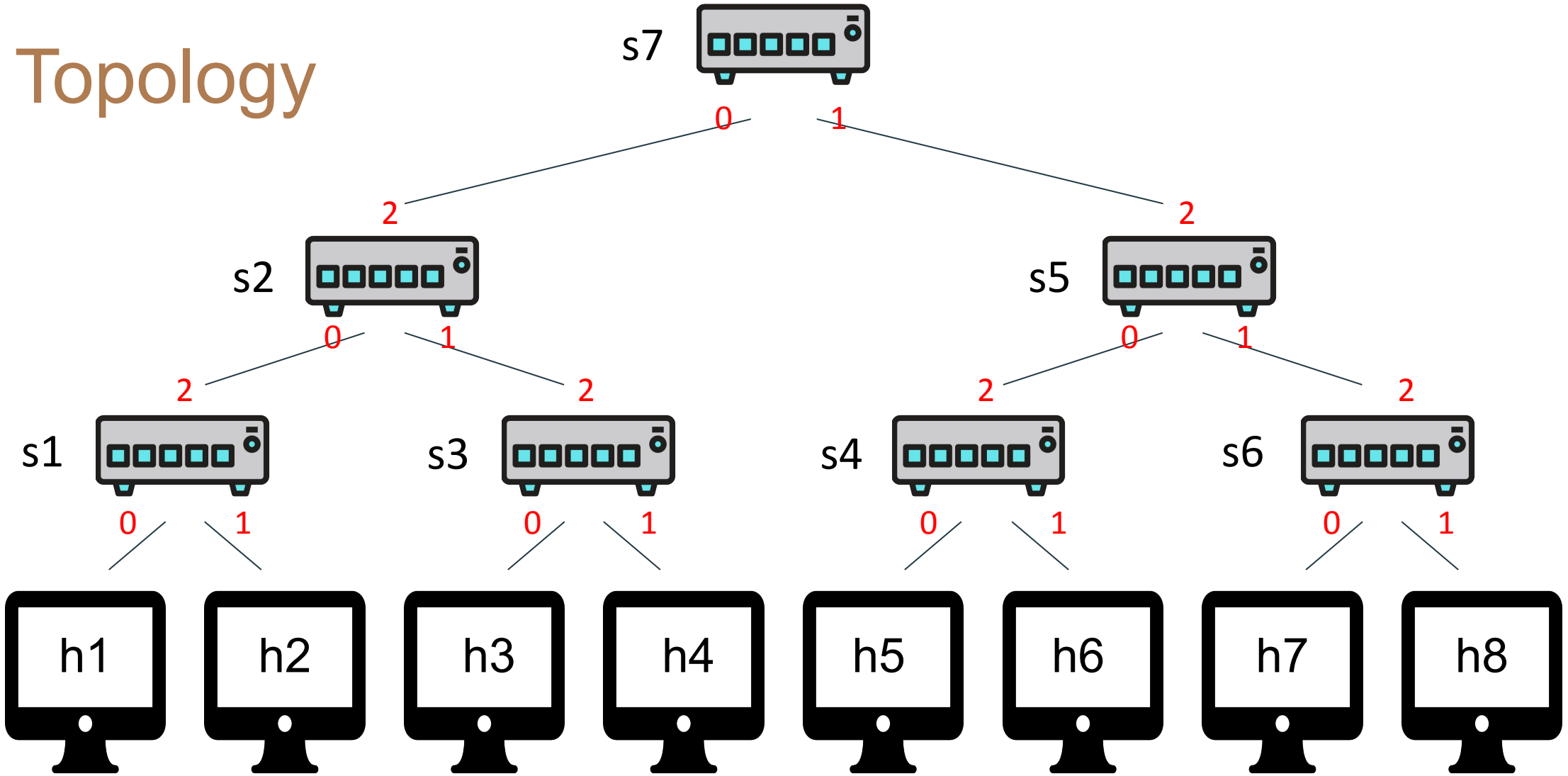
NSCAP Homework 2

Learning Switch Protocol and ARP Protocol

Objective

- Learn how the ARP table and MAC table work
- Write an interactive python program like what the mininet does
  - Class Host
    - IP, mac, arp_table, …
    - Functions: handle_packet, send, update_arp, …
  - Class Switch
    - mac_table
    - Functions: handle_packet, send, update_mac, …

Topology


s7
0
1
2
2
s2
s5
0
1
0
1
2
2
2
2
s1
s3
s4
s6
0
1
0
1
0
1
0
1
h1
h2
h3
h4
h5
h6
h7
h8

Topology

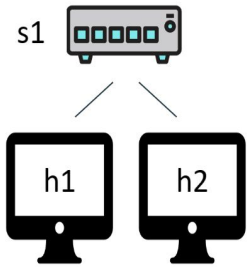
- Import setting.py
  - get_hosts() returns hosts' names
  - get_switches() returns switches' names
  - get_ip() returns a dictionary
    - key: host_name
    - value: its IP address

```
def get_hosts():
    command = 'h1 h2 h3 h4'
    return command
def get_switches():
    command = 's1 s2 s3'
    return command
def get_ip():
    ip_dict = dict()
    ip_dict["h1"] = 'h1ip'
    ip_dict["h2"] = 'h2ip'
    ip_dict["h3"] = 'h3ip'
    ip_dict["h4"] = 'h4ip'
    return ip_dict
```

Four hosts and three switches in this example

Topology

- Import setting.py
  - get_mac() returns a dictionary
    - key: host_name
    - value: its MAC address
  - get_links() returns all links
    - links split by “ “
    - The two nodes of a link are split by “,”
    - e.g.,
      - ‘h1,s1 h2,s1’ means




```python
def get_mac():
    mac_dict = dict()
    mac_dict["h1"] = 'h1mac'
    mac_dict["h2"] = 'h2mac'
    mac_dict["h3"] = 'h3mac'
    mac_dict["h4"] = 'h4mac'
    return mac_dict

def get_links():
    command = 'h1,s1 h2,s1 h3,s3 h4,s3 s1,s2 s2,s3'
    return command
```

Command

- ping
  - No need to print anything
  - e.g.,
    - h1 ping h2
- show_table
  - show arp_table or mac_table
  - show_table {host-name/switch-name}
  - show_table {all_hosts/all_switches}

Command

- clear
  - clear {host-name/switch_name}
- If the entered command is not “ping,” “show_table,” or “clear”
  - Print “a wrong command”

Example

- 4 hosts + 3 switches

```
$ python3 main.py
>> show_table h1
ip : mac
---------------h1
>> show_table all_hosts
ip : mac
---------------h1:
---------------h2:
---------------h3:
---------------h4:

>> show_table all_switches
mac : port
---------------s1:
---------------s2:
---------------s3:

>>
```

```
>> h1 ping h4
>> show_table h1
ip : mac
---------------h1
h4ip : h4mac
>> show_table all_hosts
ip : mac
---------------h1:
h4ip : h4mac
---------------h2:
---------------h3:
---------------h4:
h1ip : h1mac

>> show_table all_switches
mac : port
---------------s1:
h1mac : 0
h4mac : 2
---------------s2:
h1mac : 0
h4mac : 1
---------------s3:
h1mac : 2
h4mac : 1

>>
```

Example

- 4 hosts + 3 switches

```
>> clear s1
>> show_table all_switches
mac : port
---------------s1:
---------------s2:
h1mac : 0
h4mac : 1
---------------s3:
h1mac : 2
h4mac : 1

>> clear
a wrong command
>>
```

Format: clear {host-name/switch-name}

Sample code

```
class host:
    def __init__(self, name, ip, mac):
        self.name = name
        self.ip = ip
        self.mac = mac
        self.port_to = None
        self.arp_table = dict()
        # maps IP addresses to MAC addresses
    def add(self, node):
        self.port_to = node
    def show_table(self):
        # display ARP table entries for this host
    def clear(self):
        # clear ARP table entries for this host
    def update_arp(self, ...):
        # update ARP table with a new entry
    def handle_packet(self, ...): # handle incoming packets
        # ...
    def ping(self, dst_ip, ...): # handle a ping request
        # ...
    def send(self, ...):
        node = self.port_to # get node connected to this host
        node.handle_packet(...) # send packet to the connected node
```

Sample code

```
class switch:
    def __init__(self, name, port_n):
        self.name = name
        self.mac_table = dict()
        # maps MAC addresses to port numbers
        self.port_n = port_n # number of ports on this switch
        self.port_to = list()
    def add(self, node): # link with other hosts or switches
        self.port_to.append(node)
    def show_table(self):
        # display MAC table entries for this switch
    def clear(self):
        # clear MAC table entries for this switch
    def update_mac(self, ...):
        # update MAC table with a new entry
    def send(self, idx, ...): # send to the specified port
        node = self.port_to[idx]
        node.handle_packet(...)
    def handle_packet(self, ...): # handle incoming packets
        # ...
```

Demo

- Run some test commands
- Show your code and explain in detail how it works
  - e.g.,
    - What's the difference between broadcasting and flooding?
    - What will happen when `h1 ping h7`
  - Then, after we `clear s5` and `h7 ping h1`, what will happen?

Grading Policy

- Demo
  - Correctness of the output: 30%
    - After the first “ping” (e.g., `h1 ping h4`)
      - mac_table: 10.5% (each switch: 1.5%)
      - arp_table: 9% (h1 and h4: 3%, each of other hosts: 0.5%)
    - After “clear” and the second ping
      - mac_table: 10.5% (each switch: 1.5%)

Grading Policy

- Demo
  - Explain the internal behavior of your program: Five scenarios 50%
    - Each scenario: 10%
      - Correctly orally describe the operations of the hosts and switches: 5%
      - Your code correctly implement these operations: 5%
  - Answer questions for a wrong configuration of the topology : 20% (No need to show your code.)
    - Correctly describe what problem will happen (10%)
    - Propose a solution to solve this problem (10%)

Submission

- {student_id}_hw2.zip
  - {student_id}.py